COMP0005 Group Coursework Report

Written by Ce Cao, Phoenix Sun, Utku Bilgi, Yue Wang

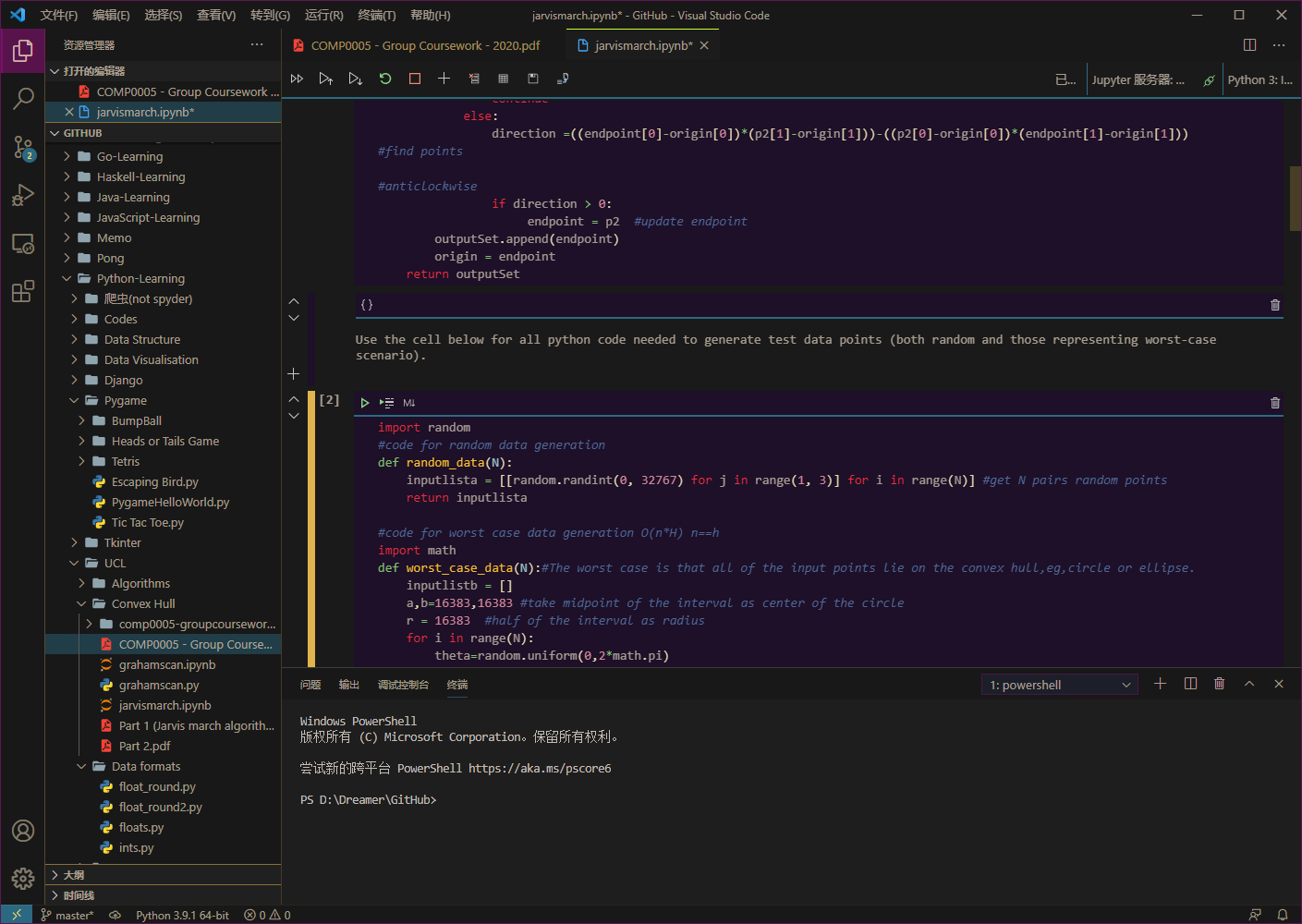
**Jarvis march algorithm**

Computational Complexity:

**Random input case:**

Theoretically: The total run time is O(Nh), where N is the size of input data, h is the size of the output (the subset of points that lie on the convex hull). h ≤ N

O(Nh), h depends on how the function ' random \_data' define.



Experimentally:

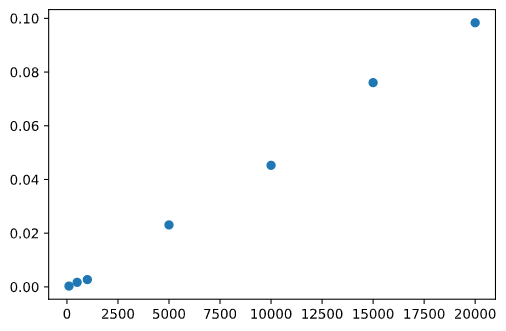
N = 100, 500, 1000, 5000, 10000,15000,20000

h = 10,15,19,25,19,20,32

(note: We run the test 100 times with each N to reduce errors, and divide each total time of N by 100 to get average execution time)

execution time (s) = [0.0003140000001167209, 0.0017018999999436346, 0.00272610000001805, 0.023069800000030227, 0.04527290000009998, 0.07606239999995523, 0.09836279999990438]

Scatter plots (x-axis is N, y-axis is the measured execution time):



Conclusion:

With the range of x-y scale of input points unchanged ([0,32767]) , the value of h does not change much when N takes 100,500,1000,50000,10000,15000,20000. The value of h is much less than N, and h is not proportional to N.

**Worst case:**

Theoretically: h = N →O(N^2)

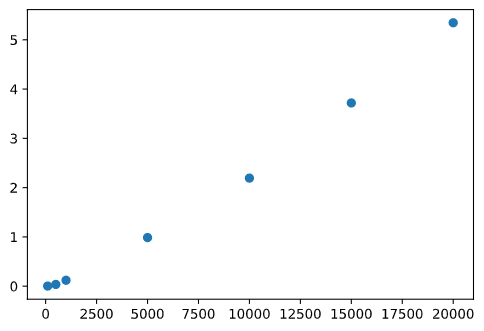
Experimentally: ' h = N ' means all of the input points lie on the convex hull. The code used here is an example which is a circle with center (32767/2 ,32767/2).

'input' N = 100,500,1000,5000,10000,15000,20000

Note: The actual value of N is always less than the 'input ' N. This is because we use ' random ' . It causes the larger 'input' N , the more repetitions. But in the actual algorithm, there is no repeated point by default.

execution time(s) = [0.002981100000170045, 0.03568360000008397, 0.12046559999998863, 0.9870078999999805, 2.193548299999975, 3.7183849999998984, 5.345930099999805]

Scatter plots ( x-axis is N , y-axis is the measured execution time):



**Graham Scan algorithm**

Computational Complexity:

**Random input case:**

Theoretically**:**

There are 3 phases of this algorithm.

Phase 1 is getting the point p0 with minimum y-coordinate, or the leftmost such point in case of a tie. This takes time O(N).

Phase 2 is sorting the point by polar angle in counterclockwise order around p0, this should take time O(NlogN).

Phase 3 goes through every point, each point is pushed in the stack exactly once, and popped at most once, so this gives us O(N) time complexity.

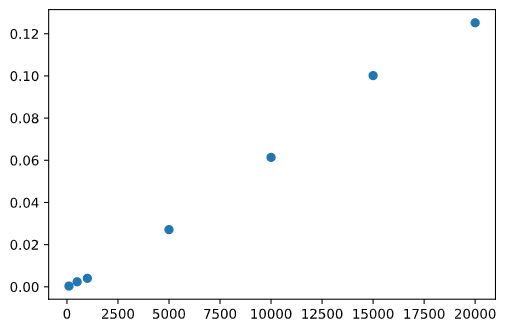
The total run time is O(NlogN).

Experimentally:

N = 100, 500, 1000, 5000, 10000,15000,20000

execution time (s) = [0.0003669000000172673, 0.0023853000000144675, 0.004041199999960554, 0.027114400000073147, 0.06137509999996382, 0.10015129999999317, 0.1252054000000271]

Scatter plots (x-axis is N, y-axis is the measured execution time):



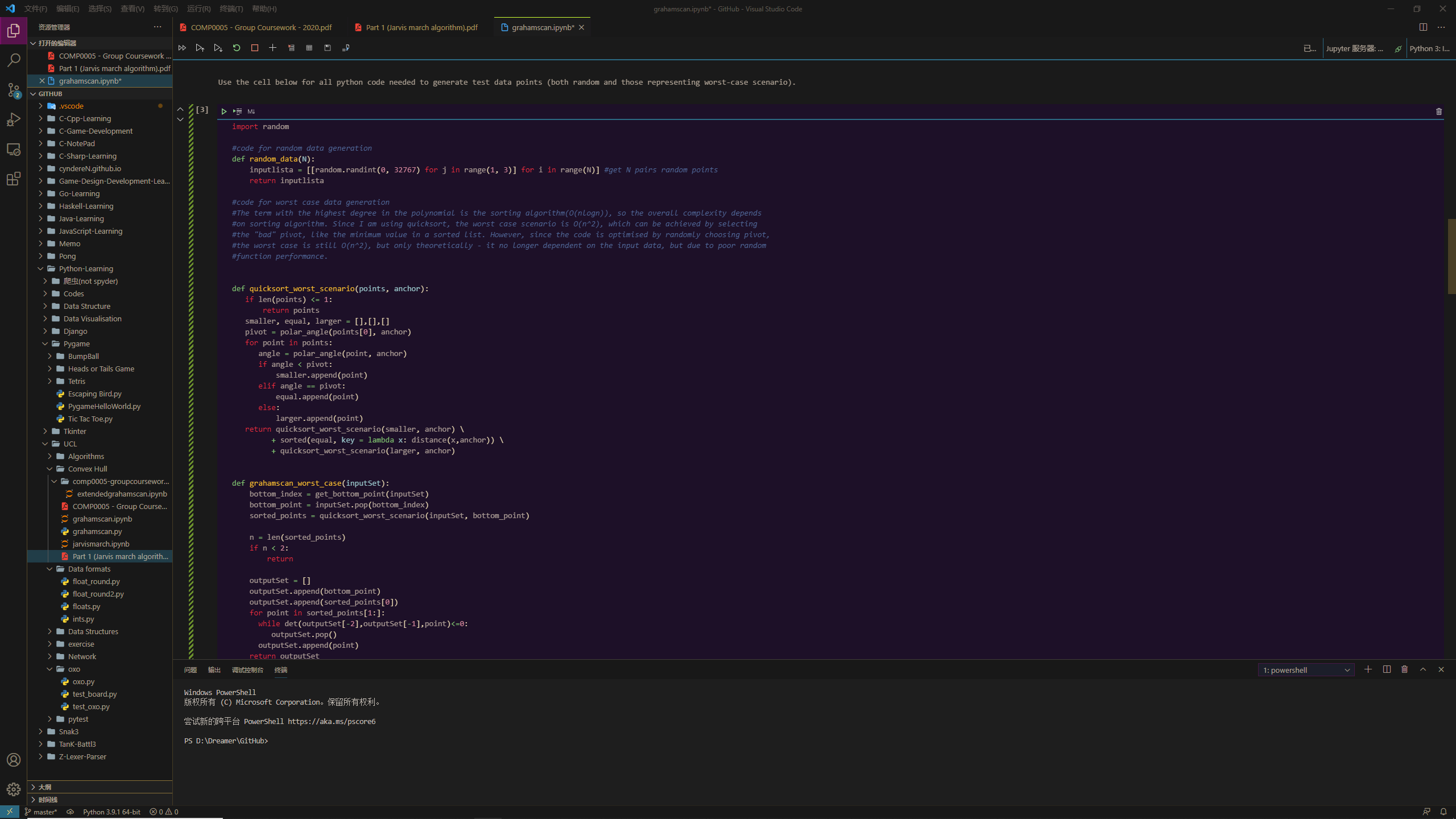
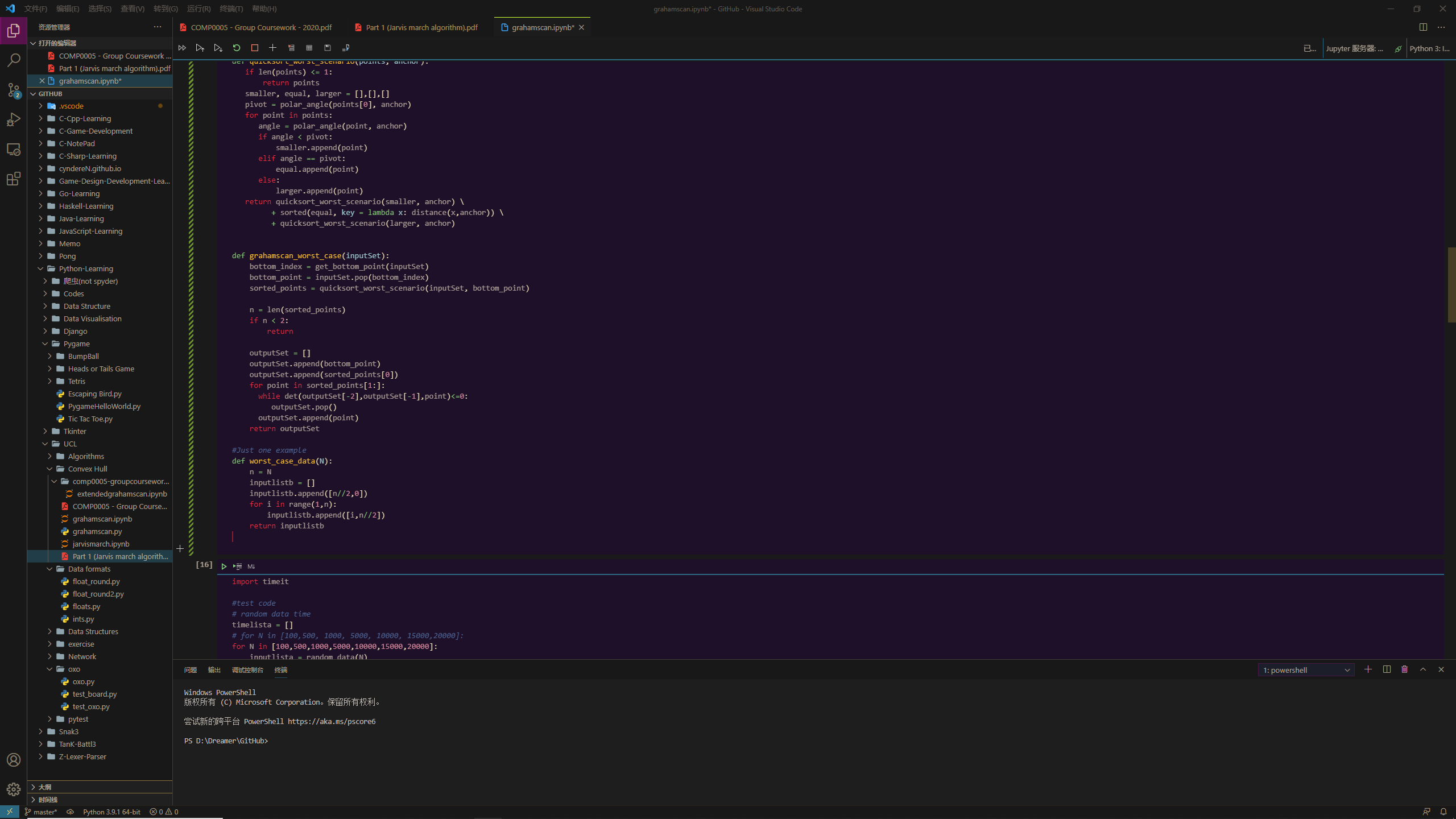
**Worst case:**

Theoretically:

Since the degree of the polynomial is determined by the term with the highest degree in the polynomial, the worst case of graham scan algorithm is determined by the worst case of sorting algorithm which has complexity of O(NlogN). The sort algorithm we use is quicksort, the worst-case scenario of quicksort is O(N^2) which can be achieved by selecting the "bad" pivot every time, like the minimum value in a sorted list. However, we enhanced quicksort algorithm in this problem with optimization of randomly choosing the pivot, this already ensures the theoretically best performance – it’s highly unlikely for the machine-determined random function to encounter with the corresponding input set that gives us worst performance. So theoretically the complexity of worst case is still O(Nlogn); The O(N^2) case is reachable, but with the possibility of 0.

Experimentally:

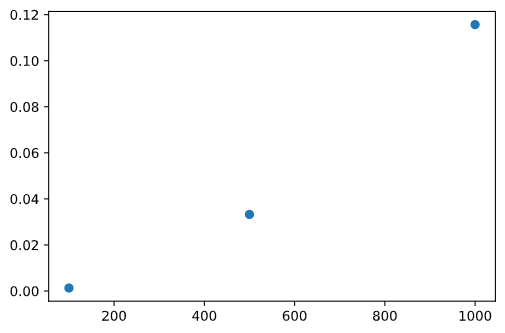
Since in most situation it’s still O(NlogN), it won’t make much difference than previous. However, we manually create a situation that has O(N^2) complexity, by making the inputSet already sorted and the quick sort algorithm choosing the “bad pivot” every time.



N = 100, 500, 1000, 5000, 10000,15000,20000

execution time (s) = [0.0012981000000991116, 0.03325760000006994, 0.11567089999994096, error, error, error, error]

Due to the limitation of maximum recursion depth in python [1] which has the number of 1000, RecursionErrors will be reported for test sets with more than 1000 points because quick sort is a recursive approach – in this worst case, it calls the stack n times if there’s n points in the test set. But we can see in the graph that the time worst case with 1000 points is nearly equal to average case with 15000 inputs.



**Extended Graham Scan algorithm**

Computational Complexity:

**Random input case:**

Theoretically: There are 3 phases to this algorithm.

**Phase 1:** Determining 4 points with minimum and maximum x and y coordinates. These 4 points must belong to the convex hull and the points inside the formed quadrilateral will get discarded. This phase has O(N) time complexity.

Diagram

Description automatically generated**Phase 2:** Sort the remaining points on their x-coordinate. Sort it on ascending order for regions 1 and 2 and in descending order for regions 3 and 4 (See Fig.1.) [2]. This phase has O(NlogN) time complexity.

**Phase 3:** Finds the convex path by considering three further points in the chosen region and by calculating the S value (cross-product of two vectors) it determines if the next point belongs to convex hull. If S is greater than or equal to zero, the next point is added to the (data structure) else the point is deleted. This phase has O(h) time complexity where h is the number of points outside the quadrilateral.

The overall time complexity is Nlog(h). Fig. 1.

Experimentally:

Need the execution times.

Scatter plots (x-axis is N , y-axis is the measured execution time):

Chart

Description automatically generatedChart, scatter chart

Description automatically generated

**Worst Case:**

Theoretically:

The worst case of the extended graham scan algorithm is determined by the worst case of the sorting algorithm (quicksort) since phase 1 and phase 3 have a running time proportional to the number of points (O(N) or O(h)). The worst-case scenario is reached when all the points lie on a straight line so the phase 1 won’t be able to discard any points because there won’t be any points inside the quadrilateral. The overall time complexity would be Nlog(N).

Experimentally:

(Edit here to explain why we didn’t do the worst case experimentally).

Improvements:

When comparing with the original Graham Scan algorithm, our extended version proves to be faster especially for large values of N. This is because in most cases at least half of the points get discarded in phase 1. Also, the cross-product calculation is significantly faster and more inexpensive than calculating polar angles in the Graham Scan algorithm. As it can be seen from the data, we collected the extended version is quicker than the original algorithm both for their average and worst case.

Reference list:

[1] “Python 3.9.2 documentation”, Python Software Foundation. (2021). Available from: <https://docs.python.org/3.9/>

[2] Selim, G. (1977). *A Fast Convex Hull Algorithm.*Available at: http://www-cgrl.cs.mcgill.ca/~godfried/publications/fast.convex.hull.algorithm.pdf (Accessed: 4 March 2021).